State-of-the-art User Interfaces for Building Operating Systems

Huiwen Xu
Institute of Applied Informatics and Formal Description Methods
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
xu@kit.edu

Hartmut Schmeck
Institute of Applied Informatics and Formal Description Methods
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
schmeck@kit.edu

***Abstract*—Being equipped with an appropriate user interface, building operating systems are capable of providing transparent information on energy consumption and generation in the household and achieving interaction with residents to implement energy management and optimization. In order to help provide a reference for the design of a more comprehensive user interface for building operating systems in the future, this paper reviews state-of-the-art user interfaces for various building operating systems and evaluates them from perspectives of a variety of use cases relating to smart buildings as well as different technical characteristics about successful user interfaces.**

Keywords*—*Building operating system; User interface; Energy management; Smart building; Smart home; Visualization

I. Introduction

With the rising share of electricity from fluctuating, decentralized, and hardly controllable renewable sources, balancing power demand and supply in a short time is a big challenge that is needed to be addressed. In addition, the gradual depletion of fossil fuels and the deteriorating climate demand an increasing need for improving energy efficiency in household buildings since in Europe household energy consumption accounts for about 25% of the total energy consumption (according to Eurostat 2014 [1]). Under such circumstances, a rising number of building operating systems (BOS) are springing up, which can make normal buildings controllable and smart by knitting together all kinds of actors, e.g. appliances, sensors, smart plugs, meters, etc. in the building into one single system. Within a BOS, actors can exchange information and cooperate with each other to achieve common goals. Several papers (e.g. [2], [3], [4] and [5]) have focused on the design and implementation of various building operating systems. An indispensable component of a BOS is the user interface, which plays an essential role in interacting with users, since building operating systems cannot work properly without user involvement. They need to be under control of users and inform them about system states in time, e.g. by providing information about local energy use. Since goals and functionalities provided by different building operating systems vary, their user interfaces are also very different. A first step in improving user interfaces would be a careful review and evaluation of state-of-the-art user interfaces for current building operating systems. Since we are not aware of such comparative evaluations, this is the major contribution of this paper.

After reviewing various user interfaces for building operating systems in Section II, a collection of use cases related to smart buildings is proposed in Section III. The user interfaces are further evaluated in Section IV with respect to the proposed use cases as well as a series of technical characteristics, respectively. The paper concludes with a summary and an outlook on the design of future user interfaces for building operating systems.

II. Review of User Interfaces for Building Operating Systems

This section introduces user interfaces for a number of popular building operating systems and analyzes their underlying functionality. These user interfaces are selected based on two criteria. Firstly, they are open source. Secondly, their use cases are related to smart home or building or to home or building automation.

A. EF-Pi

EF-Pi (Flexibility Platform and Interface) [6] has previously been known as FPAI (Flexible Power Application Infrastructure). Its mission is to provide an interoperable framework to bring together all sorts of appliances using different standards and protocols and to enable compatibility of heterogeneous energy management services.

The user interface of EF-Pi is a widget-based user interface framework. Widgets can be created freely and customized by developers. Normally, every energy application will come with a widget to interact with users. Device drivers may also provide their respective widget to display information about current states of the device. In the HEGRID project [7], two widgets (one for temperatures of a micro combined heat and power plant (μCHP) and another one for operational details) [8] were developed for a μCHP driver based on the EF-Pi framework.

EF-Pi gives developers full control and freedom to customize their own widgets. A widget consists of a user-defined back-end Java class which can communicate directly with the EF-Pi runtime and three front-end resources: an html-file, a JavaScript file and a cascading style sheet [9]. The Java class defines attributes and provides methods to access those attributes that will be displayed to end users on the widget. The html-file and style sheet determine how and in which style the data should be displayed. The JavaScript file is the connector

between the Java class and the html file. It periodically gets updated data from the EF-Pi runtime by invoking methods defined in the Java class and the data will be further accessed by the html-file.

The user interface provided by the EF-Pi is just a framework, which provides possibilities for developers to define their own widgets. It brings high flexibility and customizability, but meanwhile also weakens the power of the EF-Pi user interface. Since the data model provided by the EF-Pi user interface is just an empty Java class, application developers have to construct the Java class on their own, which makes every widget proprietary to its underlying device driver or application. In other words, widget developers have to design and develop their widget from scratch. There are no reusable UI modules provided by the EF-Pi user interface to help them simplify or unify widget development.

B. openHAB

OpenHAB (open Home Automation Bus) [10] is a generic home automation solution for embracing an abundance of fragmented home automation technologies into one single solution.

Currently, in its standard configuration, openHAB provides three user interfaces [11], which are the Paper UI, the Basic UI and the Classic UI. The Paper UI is an administration tool for configuring an openHAB instance. The functions supported by the Paper UI include installing or uninstalling add-ons, discovering and adding new things, configuring system, etc. However, the Paper UI can not cover all of the configuration tasks. So in some cases, it is still necessary for users to resort to textual configuration files.

Different from the Paper UI whose purpose is mainly to specify configurations, the Basic UI and the Classic UI are user interfaces meant for control and operation. These two user interfaces provide the same functions but different look and feel, since they use different layout frameworks. The weakness of the Classic UI is that it has less visual appeal, since its style is close to the style of the old iOS which does not match the modern standards. Except for visual differences, the Basic UI and the Classic UI share the same design philosophy. They use a so-called sitemap, which is a declarative UI definition, to define the layout of the UI page. A number of UI elements including among others Colorpicker, Chart, Frame, Group, and Image are provided by openHAB. They show information or status data and some of them provide options to interact with users. These elements can be flexibly incorporated into a sitemap and afterwards be rendered on the UI page. Clear and easy syntax is given to ensure elements in the sitemap are organized in an appropriate way.

In openHAB, an essential concept utilized by the automation logic and the user interface of openHAB is a so-called item, which represents either a real component e.g. Dimmer, Player, Switch, etc. or an abstract functionality e.g. Colour, Number, Group, etc. Besides, the rule engine of openHAB also allows developers to define rules to trigger automation processes. A rule may contain any number of conditions that can be triggered by some events including commands and status updates for items, special times and certain system statuses. In openHAB, items and rules are defined in different files with respective syntax.

The definition of the sitemap is stored in a text file. As mentioned above, the Paper UI is designed for system configuration. However, defining and editing the sitemap cannot be achieved via the Paper UI. Instead, openHAB provides an integrated development environment called Eclipse SmartHome Designer to operate the sitemap, which supports syntax check, content assist, etc. to define sitemaps as well as aforementioned items and rules.

In conclusion, some of the system configurations of openHAB can be achieved with the aid of the Paper UI. Direct use of textual configuration files is however still needed as a result of limited functions provided by the Paper UI. As for the system visualization and control, the Basic UI and the Classic UI come into play by parsing the sitemap file. Thus to construct a customized user interface for an openHAB instance, developers do not need any front-end programming knowledge apart from mastering the necessary syntax for the sitemap definition. At the moment, the Basic UI and the Classic UI can only be customized via modifying the sitemap file which requires users to know the syntax of the sitemap and to know well the item definitions. In the user interfaces, there are no visual elements available that allow novices or nonprofessional users to directly manipulate them to do customizations.

C. OGEMA

OGEMA (Open Gateway Energy MAnagement) [12] is an open software platform for building automation and energy management. It comes with communication drivers that hide physical realization of protocol connections and also provides mechanisms for developers to deploy customized energy management applications on the framework.

Similar to the user interface of EF-Pi, the start page of the OGEMA user interface is also composed of widgets which show simple descriptions of correspondingly installed applications or services. By clicking on any widget, a new web page for the specific application will be opened. There are no pre-defined data models for the application UI page. Application UI pages are totally taken care of by application developers, which is similar to EF-Pi.

In OGEMA, there are three different kinds of users: the natural user, the machine user and the admin user. The natural user has rights to access web resources like installed applications within their permissions. The machine user is mostly used for applications with different resource permissions. The admin user is an extended form of the natural user and has all of the permissions to manage users, resources and applications. Unlike application UI pages, the UI framework of OGEMA has already defined a range of data models for the admin user and the natural user for different purposes in its core, e.g. data models for viewing installed applications within permissions.

The demo user interface provided by OGEMA shows a number of installed applications consisting of basic switch GUI, device configurator, log data visualization, schedule viewer, graph generator, logging configuration, room climate station, KNX driver GUI, OGEMA framework administration, OGEMA start page, the reference implementation of the REST interface and security GUI. To get detailed information related to each application, one can download and run the OGEMA demokit [13].

The working mechanism of the OGEMA user interface is similar to that of the EF-Pi user interface. It gives full freedom for application developers to define the UI page for their own applications. But at the same time, it also suffers from the same weakness as the EF-Pi user interface, since it also has no predefined UI data models for the user interface of its applications, which means no modules that can be reused by application developers are available.

D. FHEM

FHEM (Friendly Home Automation and Energy Measurement) [14] is a central home automation server which supports different home automation hardware systems, e.g. FS20, HomeMatic, etc.

To facilitate users having access to FHEM, a standard built-in user interface called PGM2 [15] is provided by the FHEM installation. PGM2 is the default user interface of FHEM and implements a simple web server. Since FHEM is controlled through its predefined readable/ASCII commands, PGM2 provides a command field to allow end users to send FHEM commands directly to the back-end system in order to manage or control devices in their household, e.g. grouping devices into rooms or sending commands to devices. Any events triggered by devices will be recorded into a log file and displayed via an event monitor. In order to help users to reduce the burden of memory, PGM2 provides graphical elements which can implement functions supported by most of the FHEM commands. But for complex home automation, e.g. event-based execution, timed switching and so on, there are no graphical elements to support these functions, so users have to resort to PGM2 command editors. In addition, a floor plan module can be integrated into PGM2 to implement managing and controlling switchable devices on the floor plan of the household. To be able to change styles and adjust to devices with different screen size, PGM2 allows users to select an appropriate one from a list of available styles.

An alternative lightweight but feature-rich user interface for PGM2 is called FHEM Tablet UI [16], which is based on HTML/CSS/JavaScript and thus does not impose any additional requirements on the FHEM server. Compared to PGM2, the widget based Tablet UI is much more flexible for third-party UI developers to customize their own user interface. With the help of a drag-and-drop and multi-column jQuery grid plugin, the UI page can be divided into a number of cells which can be used as containers for organizing widgets according to different topics. Unlike PGM2, which requires users to directly send commands to the FHEM server via a command field, Tablet UI converts FHEM commands into various attributes of corresponding widgets, which are working as the API to interact with the FHEM server. Currently FHEM Tablet UI comes with more than 20 types of widgets in its standard installation, e.g. thermostat, switch, label, etc. These Widgets with different attributes can be customized and reused. But they do not directly support complex home automation.

E. OSH

OSH (Organic Smart Home) [17] is a flexible energy management framework for smart buildings which is based on a hierarchical Observer/Controller architecture in Organic Computing.

The OSH emphasizes aspects of interacting with human users and respecting their needs. Its user interface is called Energy Management Panel (EMP). Currently, two versions of EMP are available: KIT EMP [18] and FZI EMP [19], respectively. These two EMPs are used in different buildings. KIT EMP is the user interface designed for the KIT Energy Smart Home Lab (ESHL) [20]. The basic hardware setup of the ESHL is presented in [21]. FZI EMP is developed to satisfy the needs of the FZI House of Living Labs (HoLL) [22].

The functionalities provided by the KIT EMP are mainly composed of three aspects: the visualization of energy data, a floor plan based household overview and energy use predictions. For the visualization of energy data, real-time power consumption and generation in the household, energy price signals and historical energy use are displayed. To help get a complete picture of the current energy use in the household, the KIT EMP provides an overview of energy flows in the ESHL, which displays energy flows between the ESHL, power grid and four high power electrical equipment including a photovoltaic, a μCHP, an air conditioner and an electric vehicle. To check the status and energy use for each of these devices in the ESHL, KIT EMP offers a clickable floor plan for the ESHL. By clicking on different areas of the floor plan, devices located in the corresponding area along with their status and power will be displayed and they can be further controlled by turning on or off or specifying a temporal degree of freedom. In addition, predictions of the future energy use for the whole ESHL and different devices in the ESHL are also available.

Compared to the KIT EMP, the FZI EMP provides more comprehensive features. For example, the overview of energy flows is multi-modal, i.e. based on different commodities. Besides electricity, it also integrates gas, heat and cold. The floor plan provided by the FZI EMP is configurable so it is not limited to the building of FZI. Some more features like allowing users to set the thermal degree of freedom for the meeting room with the help of a calendar widget are also supported by the FZI EMP.

As the user interface of the OSH, the EMP provides transparent information on household energy consumption and generation and helps discover and specify degrees of freedom

for appliances. Although the FZI EMP makes some improvements for the KIT EMP, it still has room for enhancements. For instance, the layout of the EMP is not responsive. The configuration feature of the EMP is also not far developed. In addition, many features like overview of energy flows are proprietary for either the ESHL or HoLL. They are hard-coded and specifically designed for these two buildings, which make the EMP inflexible and hard to apply to other households.

F. smartVISU

SmartVISU [23] is a framework for the visualization of KNX home automation systems. It creates solutions for visualizing KNX-installations by using simple html pages whose contents are organized into different kinds of blocks. As a front-end framework, it can work smoothly with different home automation back-ends. The recommended back-end for smartVISU is Smarthome.py.

To visualize all kinds of device values, home status and some other information, smartVISU provides a collection of various widgets which are classified into Basic, Calendar, Device, Multimedia, Phone, Plots, Status, Time/Clock and Weather. The widgets in smartVISU are always expressed in a double quoted encapsulated string which contains a widget name and a unique ID, the item or GAD (GAD, short for Group Address, is interchangeable with the term "item" in smartVISU.) that the widget will address to and some other setting parameters.

To facilitate organizing widgets with similar attributes or in similar scenes in a clear and reasonable way, smartVISU provides five different design blocks attached with corresponding templates of html code which can be used directly by developers. Within blocks, a clear syntax can be used to add any number of devices just by inserting respective widget expressions as explained above.

Powered by SmartHome.py, smartVISU also supports advanced home automation, e.g. setting auto timers to perform specific actions or creating different scenes to automatically adjust device states to match different needs of the corresponding scene, e.g. party, dinner, etc. All these functions are implemented via operating configuration files of the SmartHome.py. Besides, smartVISU also provides six kinds of widgets to display different charts including "comfortable chart" which indicates comfortable zones with respect to relative humidity and effective temperature, and a temperature rose chart which shows all rooms with their actual and set temperatures in one diagram. Furthermore, some configurations of the system, like configurations of visual appearance, interface to the back-end, weather, phone and calendar can be achieved with smartVISU. Thanks to a responsive layout, rich features and flexible design elements, smartVISU has been accepted by many KNX-enabled home automation systems as their user interface. One aspect that limits the application of smartVISU is the small variety of widgets for complex devices. Currently, only five device widgets, including blind, codepad, dimmer, room temperature regulator and shutter are available. This is because smartVISU is specifically designed for the visualization of KNX home automation systems.

G. HomeGenie

HomeGenie [24] is an open-source home automation server that can integrate devices, services and a number of currently popular communication standards into a common home automation environment. For data visualization, it is equipped with a built-in web user interface [25] which is customizable, configurable and extensible. To this end, HomeGenie comes up with a series of design measures and implementation tools.

In HomeGenie, all devices and services are abstracted into so-called modules, whose parameters indicate corresponding features or states of the devices or services. Each module is identified by Domain, Address, Type and Widget. The Domain is used to divide all modules into different groups according to protocols or other features. The Address refers to the identifier of a module in its Domain. The Type indicates what kind of devices or services the module belongs to. The Widget displays status information and some control items about the module in the user interface.

There are already a number of widgets available in HomeGenie. However, the existing widgets may still be customized. Besides, users are also provided with possibilities to create their own widgets. The user interface of HomeGenie comes with a widget editor to implement the widget customization and a program editor to implement the business logic of the module. HomeGenie utilizes automation programs which can be coded using different programming languages in the program editor to implement and extend system functionalities. Plenty of helper classes are available to ease programming. Each automation program associates one or multiple modules so that it can display data in the bound widget of the modules and what is more, interact with users by adding program options and program features to corresponding modules, that will be further displayed as option fields in the user interface from which users can configure parameters. Besides, HomeGenie also allows users to create different scenarios which can be executed either manually or automatically to satisfy different needs, e.g. sunrise colors scenario or group lights on scenario, etc. HomeGenie provides an option of Record Macro [26] to create scenarios. By recording macros, all performed commands will be recorded into Wizard scripts, which can be manually executed or can be triggered on time based events by adding conditions.

The user interface of HomeGenie is designed as a control panel for users to communicate with the HomeGenie server. By equipping it with a widget editor and a program editor, the HomeGenie user interface is flexible to be configured and customized, and with a responsive layout, it can adapt to all kinds of terminal devices.

III. USE CASES

The current market of building operating systems is pretty fragmented. A wide range of services about energy management and home automation is provided by different building

operating systems. This section proposes a series of use cases relating to smart buildings based on research that has been done, e.g. [27] and [28] as well as scenarios of existing building operating systems.

Use Case 1: Basic Home Automation

Devices in the household building can be remotely controlled via the user interface of the building operating system. Examples could be switching on or off lights, rolling down or up blinds, etc. The reason why it is called basic home automation is that appliances are simply controlled manually by residents at times of need.

Use Case 2: Advanced Home Automation

Advanced home automation is more complex than basic home automation. It allows residents to create different scenarios by executing a series of commands on devices by a single action or configure devices to automatically respond to certain events when some pre-configured conditions are satisfied.

Use Case 3: Possibilities to Specify Degrees of Freedom for Devices

The degree of freedom of household devices means the potential of re-scheduling. Some devices like the stove or the TV do not have degrees of freedom since they have to start working once residents want to use them. Devices like the washing machine or the hot-water boiler have high degrees of freedom since their working schedules can be shifted on the timeline. For schedulable devices, users can specify their degrees of freedom in the user interface based on their needs. After specifying degrees of freedom, the building operating system could make use of its optimization algorithms to calculate an optimized working schedule for devices.

Use Case 4: Visualization of Building-level Energy Data

Residents can view the current state or energy flows of the whole building, e.g. voltage, frequency, total power consumption and generation, etc. This use case only refers to the global building-level energy data. It is not about energy data of a single device.

Use Case 5: Visualization of Device-level Energy Consumption

Residents can view the current state as well as energy data of every single device, smart plug or sensor in the household. The devices which are viewed in this use case refer to those which are consuming power.

Use Case 6: Visualization of Device-level Energy Generation

With the increasing popularity of domestic photovoltaic panels, more and more consumers are becoming prosumers. Some buildings are also equipped with one or more μCHPs, which are cogeneration systems that can generate heat and electricity at the same time by using different energy sources. Residents can be informed of power generation of every generating equipment in their household.

Use Case 7: Visualization of External Signals

External signals like energy cost and load limit information or warning messages from utility company, demand side manager or other actors can be accessed by residents in the user interface.

Use Case 8: Role Based Access Control

To restrict system access to authorized users, residents can be classified into different roles. Each role is granted different permissions. Residents are only allowed to access the system within the scope of their permissions. For example, the OGEMA system provides three roles including admin user, natural user and machine user.

Use Case 9: Floor Plan Based Device Organization

Residents can intuitively view and manage devices based on a floor plan of the household. Since there is a visual consistency from the physical world to the floor plan, residents can easily transfer the physical location of devices in the household to the floor plan.

Use Case 10: Visualization of Historical Energy Costs

Residents can view their historical energy costs for the whole building. If generating equipments, e.g. photovoltaic panels are installed, residents can make money by selling the surplus electricity to the energy supplier. In this case residents can also view their historical profits from the self-generated electricity. By looking at historical energy costs, residents are able to get a general idea of the energy use in their household.

Use Case 11: Visualization of Historical Energy Data

Residents can view the historical data about energy consumption or generation of devices in the household. Historical energy price and load limit signals could also be provided together with energy history of the device, so that it is easier for residents to be aware of whether they have used the device in an appropriate way in the past.

Use Case 12: Prediction of In-house Energy Use

The prediction of energy consumption or generation of devices along with price and load limit signals can be visualized in the user interface. The energy use of devices with big power consumption or generation e.g. photovoltaic or μCHP can be predicted and sent to residents so that they can adjust their energy use at the appropriate time in the future to either improve the self-consumption or the self-supply or shift loads to the cheap price period.

Use Case 13: Support for System Configurations

The user interface provides residents with options to configure the underlying building operating system, e.g. loading drivers for specific appliances, adding new users, or configuring parameters for optimization algorithms used by the building operating system.

Use Case 14: Provision of Value-added Services

Except for normal services about home automation and energy management, residents also have access to some other value-added services e.g. weather information, entertainment, security service, etc. to make daily life convenient, comfortable and secure.

Use Case 15: Visualization of Historical Data for the Single Resident

In a household building, most probably there are more than one residents. Every resident can view her own historical energy data in the past. This use case could be achieved by combining with role-based access control. Each resident can be assigned to a certain role. They might need to be authenticated

before viewing their historical energy data in order to protect privacy.

Use Case 16: Integration of Electric Vehicles

The electric vehicle can be integrated into the building operating system to be used as a flexible mobile energy storage device. The use of electric vehicles has to respect residents' actual needs or wishes. To this end, the user interface has to provide options for residents to specify some parameters, e.g. the next use time, minimal mileage, etc. The building operating system has to ensure enough electricity in the battery of the electric vehicle when residents want to use it.

Use Case 17: Connection to a User Community

The building operating system in the future could possibly connect to corresponding communities to exchange information or involve in gamification or statistic calculation. As a result, residents can view their historical energy data compared with the average value of the counterpart in the user community. They may also get suggestions about energy use from residents in the same community.

Use Case 18: Support for Setting Building Optimization Goals

The building operating system allows residents via its user interface to set single or multiple building optimization goals, which could be optimizing for the lowest cost, the maximal self-consumption or self-generation, or the minimal greenhouse gas emission, etc. Different optimization goals can be taken into consideration at the same time, and an optimal solution for the energy use in a building can be provided by using multiple-criteria decision-making methodologies.

In summary, the aforementioned use cases can be regarded as the combination of available functionalities supported by the currently popular building operating systems and the suggestions of relevant researchers about smart buildings. They can be used as the functional implementation reference for the design of future user interfaces for building operating systems. On the other hand, they can also be used as evaluation criteria to evaluate the functional integrity of the existing user interfaces for building operating systems.

IV. Evaluation of User Interfaces for Building Operating Systems

Section II introduced a number of popular user interfaces for building operating systems. These user interfaces provide a wide range of functionalities and features which are different from one another to help users get insight into their in-house energy use, have access to energy data of appliances as well as the whole building and gain control over them to make their building smart and energy efficient. This section evaluates these user interfaces from two aspects. One aspect is from the use cases presented in Section III. Another aspect is based on technical characteristics of user interfaces. To this end, a list of technical evaluation criteria is proposed.

A. Use Case Based Evaluation

This section compares the user interfaces in Section II and evaluates them (cf. Table I) on the degree of support for the use cases proposed in last section.

EF-Pi and OGEMA provide only a front-end framework for integrating widgets, which has no data models predefined. The content of the widget in the user interface is open to corresponding application developers. Since it makes no sense to evaluate a user interface framework with empty widgets, the demo user interface of OGEMA participates in the evaluation. EF-Pi does not provide a demo user interface, so the μCHP widgets developed on EF-Pi framework get the evaluation. OpenHAB comes with three standard user interfaces, but this section only evaluates two of them, the paper UI and the basic UI, because another user interface, classic UI, is the same as the basic UI except for different visual effects. FHEM Tablet UI and smartVISU are also just frameworks to create visualization, so their user demos ([29] and [30]) are included in the evaluation.

The μCHP widgets of EF-Pi are specially designed for displaying energy data about the μCHP, including power generation from an Otto-engine and a heating element, so it only supports use case 5 and 6.

The paper UI of openHAB is not for control, but is used to set up and configure the openHAB instance, therefore it supports use case 13. The basic UI supports both basic and advanced home automation. Visualization of device energy data can be achieved with the aid of power consumption measuring modules. As persistence being available in openHAB, solutions to visualize time series of historical energy data in the basic UI are provided. In addition, value-added services like weather forecast and Google maps are also available to get integrated into the user interface.

The demo user interface of OGEMA comes with a few widgets. One of them is a basic switch GUI, which allows residents to view states of controllable devices and switch them, so use case 1 and 5 are supported. Besides, the user interface also supports role based access control (use case 8) and system configuration. One of the available widgets called log data visualization is able to display logged sensor readings, so use case 11 is also supported. Another widget is called OGEMA framework administration, which allows the administrator to manage applications, users and resources (use case 13).

Since FHEM's built-in user interface PGM2 provides residents with possibilities to interact with the FHEM server by directly sending commands, both basic and advanced home automation like timed switches and event notifications can be achieved. However, the tablet UI does not directly support advanced home automation, since it provides only basic widgets with corresponding attributes. Advanced home automation needs to be taken care of by developers. PGM2 supports use case 9 because it can be extended with a floor plan module. To visualize historical log files, PGM2 can directly use the Plot command to create plots, and in tablet UI, a third party chart widget can be used. What is more, PGM2 supports system configuration by sending commands to the server in the command field. Tablet UI is able to provide value-added services like weather forecast via integrated or third party widgets.

TABLE I: Evaluation results of the user interfaces in section II based on use cases in section III

Use Case \ BOS UI	EF-Pi μCHP widgets	openHAB		OGEMA Demo UI	FHEM		OSH		Smart-VISU Demo	Home-Genie UI
		Paper UI	Basic UI		PGM2	Tablet UI	KIT EMP	FZI EMP		
use case 1	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
use case 2	✗	✗	✓	✗	✓	✗	✗	✗	✓	✓
use case 3	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
use case 4	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
use case 5	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓
use case 6	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
use case 7	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
use case 8	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
use case 9	✗	✗	✗	✗	✓	✗	✓	✓	✗	✗
use case 10	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
use case 11	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
use case 12	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
use case 13	✗	✓	✗	✓	✓	✗	✗	✓	✓	✓
use case 14	✗	✗	✓	✗	✗	✓	✓	✓	✓	✓
use case 15	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
use case 16	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
use case 17	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
use case 18	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

For Organic Smart Home (OSH), its two user interfaces, KIT EMP and FZI EMP support only basic home automation. By them, residents are able to specify degrees of freedom for appliances. In addition, no matter building-level and device-level energy data as well as the in-house power generation from photovoltaic panels and a μCHP can be visualized in these two user interfaces. External signals e.g. energy price and load limit signals can also be accessed from them. They all come with a floor plan to organize devices. The difference is that the floor plan provided by the FZI EMP is configurable. Besides, historical energy data of devices can be displayed in both user interfaces. Prediction of in-house energy use is only supported by the KIT EMP. Value-added services and integration of electric vehicles are supported by both of them. Although OSH is able to optimize in-building energy use, its two user interfaces do not provide residents with options to set their optimization goals (use case 18).

By smartVISU, residents can achieve both basic and advanced home automation and view energy data of devices. It also provides different kinds of widgets to visualize historical data by creating charts with multiple series. In the demo, historical energy costs including yesterday's, the last seven days' and the last thirty days' costs can be accessed. Use case 13 is supported by smartVISU since it comes with a configuration page providing options to configure the interface to the back-end, e.g. setting driver, address, port, etc. SmartVISU also provides value-added services, e.g. weather forecast, calendar reminder, caller ID display of the phone system.

The user interface of HomeGenie supports basic home automation as well as advanced home automation like creating scenarios. In addition, it is also possible for residents to view both building-level and device-level energy data by creating corresponding modules. Use case 11 is supported by HomeGenie user interface since it can get historical data of certain modules with the aid of a Statistics module. The user interface comes with a configuration page, by which residents can configure the system, e.g. enabling control adapters and interfaces, adding groups and modules, etc. Value-added services like weather information and security alarm system are also supported by the HomeGenie user interface.

In conclusion, none of aforementioned ten user interfaces is able to support all of the use cases proposed in the previous section. Most of them support basic home automation (use case 1), visualization of device-level energy data (use case 5), visualization of historical energy data (use case 11) and provision of value-added services (use case 14). System configuration (use case 13) is supported by half of these user interfaces. Only the demo user interface of OGEMA supports role based access control (use case 8). Only KIT EMP and FZI EMP support specifying degrees of freedom for devices (use case 3), and also only these two user interfaces support integration of electric vehicles (use case 16). Only smartVISU demo supports visualization of historical energy costs (use case 10). Prediction of in-house energy use (use case 12) is supported only by the KIT EMP of Organic Smart Home. None of the user interfaces supports visualizing historical energy data for the single resident (use case 15), connecting to a user community (use case 17) and setting building optimization goals (use case 18).

B. Technical Characteristic Based Evaluation

This section collects a number of technical characteristics related to successful user interfaces. These characteristics will

be used as evaluation criteria to evaluate upon the technical implementation of the user interfaces in Section II. To make an objective evaluation, the criteria selected in this section are those that can be definitely determined, which means the evaluation result will not depend on different evaluators. Some features about good user interfaces like attractiveness or conciseness are subjective. Whether the user interfaces satisfy them varies from evaluator to evaluator, so these subjective features are excluded from this section.

- **Easy to learn and use**
  Good user interfaces are supposed to be easily and quickly learned and used by their target users. To avoid ambiguity, this section evaluates the user interfaces in Section II upon this feature based on the fact that whether the implementation of use cases in Section III requires users to have certain professional knowledge about the building operating system in their household. A user interface is evaluated as easy to learn and use if professional knowledge is not needed by users to achieve their goals, namely that users can just operate graphical elements in the user interface to achieve their goals.
- **Responsive**
  A user interface is responsive means its layout can fluidly change and respond to fit different screen sizes, which means it looks good on all kinds of terminal devices, e.g. laptops, tablets and phones by having dynamic changes of the layout, e.g. by hiding unnecessary parts to adapt differently sized screens of devices. This is an important factor to improve user experience. It helps developers to save efforts and costs for user interface development and maintenance, since there is no need for specially implementing different versions of the user interface for differently sized screens.
- **Customizable**
  A customizable user interface allows users to change its appearance and behaviour according to their preference. When it comes to customization, it could be developer oriented, which means developers have freedom to customize the user interface in terms of appearance and behavior of the user interface, e.g. modifying existing component templates. Instead of looking at developers, this feature in this section is evaluated from the user's point of view, which means users are allowed to customize the user interface based on their personal needs without leaving the user interface. Examples could be changing widget features, color schemes, font sizes and so on.
- **Modularized**
  Modularity is the degree to which a system's components may be separated and recombined [31]. Modularized user interfaces consist of a number of loosely coupled and reusable components, therefore they are more manageable and maintainable than tightly integrated user interfaces since components in those user interfaces are hard-coded, tightly knitted with each other and specially designed for a specific scenario. Usually the loosely coupled components in modularized user interfaces also provide developers with possibilities to do customization, which is corresponding to the developer oriented customization that is mentioned but not considered in the last item.
- **Consistent**
  Consistent user interfaces can help users improve efficiency and satisfaction and reduce confusion while they try to complete tasks in the user interface. To make the user interface consistent is also one of the important design principles for the user interface. Consistency can be reflected in different aspects including using the same terminology to represent the same thing, complying with generally accepted conventions, sticking to a consistent style scheme (layout, color, font, graphical elements, etc.) and so on.
- **Multilingual**
  Multilingual User Interface (MUI) originally refers to the Microsoft products that allow users to switch between different languages on a single system [32]. However the term "multilingual" in this context is not Windows-specific, but for general user interfaces. If a user interface is multilingual, end users can adapt it to various languages at runtime by switching between multiple language options provided by the user interface without the need of changing source code. In computing, this term is called internationalization and localization [33]. This feature is favorable to users who prefer different native languages.

With the aforementioned technical characteristics as evaluation criteria, the user interfaces in Section II are further evaluated, and the results can be found in Table II.

Among the user interfaces in Section II, the Basic UI of openHAB is evaluated as not easy to learn and use, since the functionality of advanced home automation provided by openHAB has to be defined as corresponding rules. To do this, users need to know syntax of the rule definition as well as knowledge of some system configurations, e.g. item definitions. FHEM's built-in user interface PGM2 is also evaluated as not easy to learn and use. The reason is that not all functionalities provided by PGM2 can be achieved via simply operating graphical elements in the user interface. Some functionalities about advanced home automation, e.g. event notification can only be achieved via sending FHEM commands to the server. In this case, users are required to master those professional FHEM commands in order to achieve certain goals, which hinders nonprofessional users from learning and using the user interface. The user interface of HomeGenie is also evaluated as not easy to learn and use because users have to resort to a widget editor and a program editor to achieve the customization of widgets and features. To this end, users are required to have programming knowledge like C#, Javascript, Python and Ruby, which is only favorable for professional users.

The Paper UI and Basic UI of openHAB, smartVISU and the user interface of HomeGenie have a responsive layout. Other user interfaces are either not responsive at all or partly

TABLE II: Evaluation results of the user interfaces in section II based on technical characteristics

BOS / UI / Characteristic	EF-Pi μCHP widgets	openHAB		OGEMA Demo UI	FHEM		OSH		Smart-VISU Demo	Home-Genie UI
		Paper UI	Basic UI		PGM2	Tablet UI	KIT EMP	FZI EMP		
Easy to learn & use	✓	✓	✗	✓	✗	✓	✓	✓	✓	✗
Responsive	✗	✓	✓	✗	✗	✗	✗	✗	✓	✓
Customizable	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓
Modularized	✗	✗	✓	✓	✗	✓	✗	✗	✓	✓
Consistent	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Multilingual	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

responsive. For instance, the EF-Pi μCHP widgets are not responsive, since contents of the widgets are just visualized by normal HTML tables. FHEM's built-in user interface PGM2 needs users to manually select different styles to adapt to different devices. The KIT EMP of OSH is partly responsive, since some components like the chart widget for displaying historical energy use of appliances can automatically reset scales when the window size changes.

Only FHEM's built-in user interface PGM2, smartVISU and the user interface of HomeGenie are customizable. PGM2 provides users with possibilities to select different displaying styles. In the configuration page of smartVISU, there are options available to change the visual appearance of smartVISU. With the user interface of HomeGenie, users are able to customize existing widgets or add new ones with the aid of a widget editor and a program editor.

The Basic UI of openHAB is modularized because there is a series of predefined Sitemap elements that can be reused by user interface developers. In OGEMA, the application pages are not modularized, since there are not any available data models. But for the role of administrator and nature user, there are some predefined data models that can be used for different purposes. FHEM Tablet UI is also modularized because it offers developers a wide variety of widget templates. Besides, smartVISU also provides a collection of various widgets that can be reused. In HomeGenie, all devices and services are abstracted into different modules which contain corresponding customizable widgets. There is already a number of widgets available in HomeGenie. So smartVISU and HomeGenie are also modularized.

All of these user interfaces are consistent, except for the OGEMA demo user interface. The reason why it is evaluated as not consistent is that the layout styles of the UI pages designed for different applications are inconsistent. Different application pages use different background colors, different styles for graphical elements, and different color schemes for highlights.

All of these user interfaces support only one language. There are no possibilities for users to adapt the user interfaces to other languages.

In conclusion, among these ten user interfaces for different building operating systems, none of them can support all of the six technical characteristics. In more detail, most of them are consistent and easy to lean and use. Half of them support modularization. Four of them have responsive layout. Only two of them are customizable, and no one provides multilingual support.

V. Conclusion and Outlook

In recent years, a rising number of local energy management applications and building operating systems are popping up. Along with these systems there are their respective user interfaces. This paper aims to offer a reference for the more comprehensive design of the user interface for building operating systems in the future. It firstly gives an overview about state-of-the-art user interfaces for different building operating systems. Subsequently, a number of use cases relating to smart buildings are proposed. In the end, evaluations of these user interfaces are made from perspectives of these smart building related use cases as well as with respect to technical characteristics. From the final evaluation results, one can see both strengths and weaknesses of these user interfaces. None of them can cover all of the evaluation criteria. They all have space to be enhanced in varying degrees from different aspects.

With the future trend of more than one building operating system serving different purposes and operating simultaneously in a building, residents might lose control over their individual targets concerning their building since different building operating systems might interfere with each other. In this situation, to make a holistic and comprehensive overview of the energy flows and a compatible control for various devices, a common user interface that can be applied to different building operating systems might be needed. Besides, according to findings, users were more satisfied with the results from the common user interface and performed better with the common interface than the integrated interface [34]. The common user interface should be generic and flexible enough to interact with users as well as communicate with heterogeneous building operating systems. So far, such common user interfaces do not exist due to certain difficulties and challenges but their design should be feasible and it is desirable to have them in the future.

REFERENCES

[1] Eurostat: consumption of energy. http://ec.europa.eu/eurostat/statistics-explained/index.php/Consumption_of_energy. [Online; Accessed: 12-06-2017].

[2] Dawson-Haggerty S., Krioukov A., Taneja J., Karandikar S., Fierro G., Kitaev N., and Culler D. E., "BOSS: Building Operating System Services," NSDI. Vol. 13, 2013, pp.443–458.

[3] Dixon C., Mahajan R., Agarwal S., Brush A. J., Lee B., Saroiu S., and Bahl P., "An operating system for the home," Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[4] Zhao P., Suryanarayanan S., and Simoes M. G., "An energy management system for building structures using a multi-agent decision-making control methodology," IEEE Transactions on Industry Applications 49.1, 2013, pp.322–330.

[5] Mauser I., Müller J., Allerding F., and Schmeck H., "Adaptive building energy management with multiple commodities and flexible evolutionary optimization," Renewable Energy 87, 2016, pp.911–921.

[6] EF-Pi. http://flexible-energy.eu/ef-pi/. [Online; Accessed: 13-06-2017].

[7] The description of HEGRID project. http://www.aifb.kit.edu/web/HEGRID/en. [Online; Accessed: 12-06-2017].

[8] Gitte C., Xu H., Rigoll F., van Eekelen J., and Kaisers M., "Multi-Commodity Energy Management Applied to Micro CHPs and Electrical Heaters in Smart Buildings," 5th DA-CH+ Energy Informatics Conference in conjunction with 7th Symposium on Communications for Energy Systems (ComForEn), 2016.

[9] Kazemier J.J., Konsman M.J., and Waaij B.D. van der, "FlexiblePower Application Infrastructure 14.10 Developer Tutorial," 2014, EIT Digital, internal, unpublished.

[10] OpenHAB. https://www.openhab.org/. [Online; Accessed: 13-06-2017].

[11] The user interfaces of OpenHAB 2. http://docs.openhab.org/tutorials/beginner/uis.html. [Online; Accessed: 12-06-2017].

[12] OGEMA. http://www.ogema.org/. [Online; Accessed: 13-06-2017].

[13] OGEMA demokit. https://www.ogema-source.net/wiki/display/OGEMA/OGEMA+Demokit. [Online; Accessed: 12-06-2017].

[14] FHEM. https://fhem.de/fhem.html. [Online; Accessed: 13-06-2017].

[15] FHEM PGM2. https://wiki.fhem.de/wiki/PGM2. [Online; Accessed: 12-06-2017].

[16] FHEM Tablet UI. https://github.com/knowthelist/fhem-tablet-ui. [Online; Accessed: 12-06-2017].

[17] Allerding F., and Schmeck H., "Organic Smart Home: Architecture for Energy Management in Intelligent Buildings," Proceedings of the 2011 Workshop on Organic Computing, OC '11, 2011, pp.67–76.

[18] Becker B., Kellerer A., and Schmeck H., "User interaction interface for energy management in smart homes," Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES, pp.1–8.

[19] Becker B., "Interaktives Gebäude-Energiemanagement," KIT Scientific Publishing, 2014.

[20] The description of Energy Smart Home Lab. http://www.aifb.kit.edu/web/Energy_Smart_Home_Lab. [Online; Accessed: 12-06-2017].

[21] Kochanneck S., Mauser I., Bohnet B., Hubschneider S., Schmeck H., Braun M., and Leibfried T., "Establishing a hardware-in-the-loop research environment with a hybrid energy storage system," Innovative Smart Grid Technologies-Asia (ISGT-Asia), 2016 IEEE.

[22] Becker B., Kern F., Lösch M., Mauser I., Schmeck H., "Building Energy Management in the FZI House of Living Labs," In: Gottwalt S., Knig L., Schmeck H. (eds) Energy Informatics. Lecture Notes in Computer Science, vol 9424, 2015, pp. 95–112, Springer International Publishing.

[23] SmartVISU. http://www.smartvisu.de/. [Online; Accessed: 13-06-2017].

[24] HomeGenie. http://www.homegenie.it/. [Online; Accessed: 13-06-2017].

[25] HomeGenie web user interface. http://genielabs.github.io/HomeGenie/#/clients. [Online; Accessed: 12-06-2017].

[26] HomeGenie scenarios and scripts. http://www.homegenie.it/docs/ scenarios.php. [Online; Accessed: 12-06-2017].

[27] Energy@home Use Cases v3.0. http://www.energy-home.it/SitePages/Activities/ Download.aspx?RootFolder=Documents/Technical%20Specifications. [Online; Accessed: 15-03-2017].

[28] ACCIONA: Smart Buildings scenario definition. http://www.fi-ppp-finseny.eu/wp-content/uploads/ 2012/05/D4.1_Smart-Buildings-scenario-definition_v1.1.pdf, 2011. [Online; accessed 08-03-2017].

[29] The user demo of FHEM Tablet UI. https://github.com/ovibox/fhem-ftui-user-demos. [Online; Accessed: 12-06-2017].

[30] The user demo of smartVISU. http://demo.smartvisu.de/index.php. [Online; Accessed: 12-06-2017].

[31] Modularity on Wikipedia. https://en.wikipedia.org/wiki/Modularity. [Online; Accessed: 11-03-2017].

[32] Multilingual User Interface on Wikipedia. https://en.wikipedia.org/wiki/Multilingual_User_Interface. [Online; Accessed: 16-03-2017].

[33] Internationalization and localization on Wikipedia. https://en.wikipedia.org/wiki/Internationalization_and_localization. [Online; Accessed: 15-03-2017].

[34] Hariri N., and Norouzi Y., "Determining evaluation criteria for digital libraries' user interface: a review," The Electronic Library 29.5, 2011, pp.698–722.